# Decorator

בשיעורים הקודמים ראינו שפונקציות בפייתון הן אובייקטים ממעלה ראשונה- אפשר להעביר אותן כארגומנט לפונקציה ,להפעיל אותן בתוך הפונקציה שקיבלה אותה, לשמור במשתנה, ולהחזיר אותן מפונקציה:

def high\_ord\_func(x: int , func: func) -> int:

    return func(x)+x

func = lambda a : a\*\*2

print(high\_ord\_func(2, func))

print(type(func))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
6  
<class 'function'>

עוד משהו שניתן לעשות , משום שפונקציות הן אובייקט בפני עצמו ניתן להגדיר פונקציה בתוך פונקציה אחרת:

def out\_f():

    def in\_f():

        print("I'm in the inner function")

    return in\_f

f = out\_f()

f()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
I'm in the inner function

עכשיו כשיישרנו קו נראה את הדוגמא הבאה:

def outer\_function(func: func):

    def wrapper():

        print("started")

        func()

        print("ended")

    return wrapper

def a\_function():

    print("I'm a function")

יצרנו פונקציה שמקבלת כפרמטר פונקציה אחרת ועוטפת אותה ב"פונקציית מעטפת" ומחזירה את הפונקציית מעטפת.  
יש שתי דרכים מקובלות להפעיל את הפונקציה:

outer\_function(a\_function)()  
#or:  
f\_wrapper = outer\_function(a\_function)

f\_wrapper()

מה שעשינו עכשיו הוא הבסיס לתבנית העיצוב קשטן על פונקציות- לקחנו אובייקט קיים והוספנו לו התנהגויות מבלי לשנות אותו.  
בפייתון יש עוד אפשרות להפעיל את הפונקציה לעיל מבלי להכניס אותה לתוך פונקציה אחרת, וזה ע"י הגדרת הפונקציה העוטפת עם @ ושם הפונקציה:

@outer\_function

def a\_function():

    print("I'm a function")

a\_function()  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
started   
I'm a function   
ended

מה שקורה מתחת לפני השטח הוא שלקחנו את a\_function ושמנו לה את הערך של outer\_function(a\_function)   
כלומר : a\_function= outer\_function(a\_function), וכל זה מבלי בצע את ההשמה בפועל.   
ובכך הרחבנו את ההנהגות של הפונקציה מבלי לשנות את ערכה המקורי.  
  
בעיה שיכולה לעלות היא מה קורה אם הפונקציה המקורית מקבלת ארגומנטים? אם ננסה להפעיל למשל את outer\_function על פונקציה שיש לה פרמטרים האם נקבל שגיאה? והתשובה היא ככל הנראה, כי כאשר אנחנו מפעילים את decoration על פונקציה אנחנו משנים את המצביע של הפונקציה (השם שלה) לפונקציית המעטפת שמתקבלת, ואם פונקציית המעטפת לא מקבלת פרמטרים אז היא תזרוק שגיאה,

@outer\_function

def param\_function(num: int):

    print(f"{num} is a number")

param\_function(3)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**---------------------------------------------------------------------------**   
**TypeError**   
Traceback (most recent call last)  **<ipython-input-32-380132969f8b>** in <module>   
 2 **def** param\_function**(**num**:** int**):**   
 3 print**(f"{num} is a number")**   
**----> 4** param\_function**(3)**  **TypeError**: wrapper() takes 0 positional arguments but 1 was given

כדי לתקן את זה נצטרך לשנות את הפונקציה העוטפת.  
משום שהיא זו שמוחזרת מהפונקציה אנחנו רוצים שגם לה יהיה את האפשרות לקבל פרמטרים.  
אבל אנחנו לא יודעים כמה ארגומנטים , אם בכלל, הפונקציה אמורה לקבל, אז איך נגדיר פונקציית מעטפת שאמורה לקבל פרמטרים אם אנחנו לא יודעים כמה ארגומנטים הפונקציה שאותה היא עוטפת אמורה לקבל?   
אז זהו שיש כלי שימושי מאוד שגם אותו ראינו בשיעורים הקודמים, והוא אופרטור כוכבית, או בשמו המקצועי \*args \*\*kwargs , תזכורת: הפרמטר '\*' מאפשר לנו להגדיר בחתימת הפונקציה מספר בלתי מוגבל של פרמטרים שיכנסו ל-tuple והפרמטר '\*\*' עוטף את הארגומנטים במילון בתנאי שהם מתקבלים צורה של מפתח וערך:

>>> def my\_sum(\*args):  
... result = 0  
... for x in args:  
... result += x  
... return result  
...  
>>> list1 = [1, 2, 3, 4, 5]  
>>> print(my\_sum(list))  
15  
>>> def f(\*\*kwargs):  
... print(kwargs)  
... print(type(kwargs))  
... for key, val in kwargs.items():  
... print(key, '->', val)  
...  
>>> f(e=1, r=2,j=3)  
{'e': 1, 'r': 2, 'j': 3}  
<class 'dict'>  
e -> 1  
r -> 2  
j -> 3

אז אם נגדיר את חתימת הפונקציה של הפונקציה העוטפת עם \*args \*\*kwargs , ניתן לפונקציה אפשרות לקבל מספר בלתי מוגבל של ארגומנטים, כך הפונקציה תוכל לקבל לתוכה כל סוג של אובייקט אפשרי:

def outer\_function(func: func):

    def wrapper(\*args , \*\*kwargs):

        print("started")

        func(\*args, \*\*kwargs)

        print("ended")

    return wrapper

@outer\_function

def param\_function(num: int):

    print(f"{num} is a number")

param\_function(3)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
started   
3 is a number   
ended

טקסט